Journal of Statistical Software

April 2011, Volume 40, Issue 10. http://www.jstatsoft.org/

spa: Semi-Supervised Semi-Parametric Graph-Based Estimation in R

Mark Culp
West Virginia University

Abstract

In this paper, we present an R package that combines feature-based (X) data and graph-based (G) data for prediction of the response Y. In this particular case, Y is observed for a subset of the observations (labeled) and missing for the remainder (unlabeled). We examine an approach for fitting $\hat{Y} = X\hat{\beta} + \hat{f}(G)$ where $\hat{\beta}$ is a coefficient vector and $\hat{f}$ is a function over the vertices of the graph. The procedure is semi-supervised in nature (trained on the labeled and unlabeled sets), requiring iterative algorithms for fitting this estimate. The package provides several key functions for fitting and evaluating an estimator of this type. The package is illustrated on a text analysis data set, where the observations are text documents (papers), the response is the category of paper (either applied or theoretical statistics), the X information is the name of the journal in which the paper resides, and the graph is a co-citation network, with each vertex an observation and each edge the number of times that the two papers cite a common paper. An application involving classification of protein location using a protein interaction graph and an application involving classification on a manifold with part of the feature data converted to a graph are also presented.

Keywords: semi-supervised learning, graph-based classification, semi-parametric models, R.

1. Introduction

In this work, we present a package geared towards providing a semi-supervised framework for processing semi-parametric models. In this setting, the data are partitioned into two sets, one is a feature set X and the other is either an additional feature set Z or observed graph G. Here X is an $n \times p$ data matrix, Z is an $n \times q$ data matrix, and the observed graph G can be written as $G = (V, E)$, where the nodes in V correspond to the observations and E is the set of similarity edges between observations. We have a labeled response for a subset of the observations, denoted by Y_L of length $m \leq n$. The data matrix X can be partitioned into

$X = [X_L^\top, X_U^\top]$, such that $X_L^\top$ corresponds to the cases for which the response is labeled. The response is missing (unlabeled) for the cases in $X_U^\top$, and we denote these missing values as Y_U. An analogous partition can be made for an additional data matrix Z. For the observed graph G, the adjacency matrix can be partitioned as follows,

$$A = \begin{pmatrix} A_{LL} & A_{LU} \\ A_{UL} & A_{UU} \end{pmatrix},$$

where A_{LL} are the labeled-labeled edges, $A_{UL} = A_{LU}^\top$ are the labeled-unlabeled edges, and A_{UU} are the unlabeled-unlabeled edges. For these data we assume that the $E[Y] = X\beta + f(G)$ for some true β and f. The objective is to use all of the data, including those cases with a missing response, in conjunction with the labeled responses Y_L to obtain an estimator of the form $X\hat{\beta} + \hat{f}(G)$.

There are several approaches in existence to perform the difficult task of semi-supervised classification with graphs, including mini-cut algorithms (Blum and Chawla 2001; Kondor and Lafferty 2002), directed graph algorithms (Eppstein, Patterson, and Yao 1997; Zhou, Schölkopf, and Hofmann 2005), graph regularization algorithms (Belkin, Matveeva, and Niyogi 2004; Joachims 2003; Zhu, Ghahramani, and Lafferty 2003), and graph smoothers (Culp and Michailidis 2008a; Nui, Ji, and Tan 2005; Wang and Zhang 2006). The natural approach to do this is loss function optimization via regularization, where here the smoothness of the function is controlled with respect to the discrete combinatorial Laplacian operator on the graph (Zhu 2008). The problem of combining X information with a graph G (either observed or constructed from Z) is often referred to as graph-based co-training (Joachims 2003; Culp and Michailidis 2008a). Existing semi-supervised graph-based approaches for the objective require one to first convert the Z data to a graph, $Z \to G[Z]$ and then set $G_F = G[Z] \cup G$, or $G_F = G[Z] \cap G$ (Culp and Michailidis 2008a). The next step is to employ one of the above graph-based semi-supervised classification approaches directly to G_F. The proposed approach implements both the fitting the fits algorithm (FTF) in (Culp and Michailidis 2008b) and the more general sequential predictions algorithm (SPA) (Culp and Michailidis 2008a), both of which perform graph-based classification with the additional flexibility of processing the X data on its original scale. This is most desirable for X data that are not continuous.

The semi-supervised **spa** package is designed to fit $\hat{Y} = \phi$ such that the form of ϕ is:

$$\phi(X, G) = X\hat{\beta} + \hat{f}(G)$$

where $\hat{f}$ is a function over the vertices of G and $\hat{\beta}$ is $p \times 1$ coefficient vector. The challenges for this approach include simultaneous semi-supervised estimation of a coefficient vector and a function over the vertices of G. Applications for an estimate of this form occur in text analysis, where the observations in the data are published text documents and the response is the document's topic. In this case a graph describing pairwise co-citations between documents is observed directly (the edge weight is the number of times that two papers reference the same documents). In addition the journal in which the document was published is available as feature information X. Therefore, we would wish to estimate $\phi = X\hat{\beta} + \hat{f}(G)$ to combine these two sources of information for classification. Several other applications of this research are quite common in web analysis (Blum and Mitchell 1998), email (Koprinska, Poon, Clark, and Chan 2007), proteomics (Kui, Zhang, Mehta, Chen, and Sun 2002; Ruepp, Zollner, Maier, Albermann, Hani, Mokrejs, Tetko, Guldener, Mannhaupt, Munsterkotter, and Mewes 2004),

genomics and drug discovery. In addition to observed graphs, this work is also applicable as an extension of standard semi-parametric modeling, where an additional feature set Z is observed separately from X. In the classical semi-parametric setting one would commonly fit $\phi(X_L, Z_L) = X_L\hat{\beta} + \hat{f}(Z_L)$ where f is a smooth function. For the **spa** package in this case we provide several R functions to convert Z to a proximity graph, $Z \to G[Z]$ and then subsequently fit $\phi(X, G[Z]) = X\hat{\beta} + f(G[Z])$. The advantage is that the information in X_U and Z_U influence training of ϕ, that is, the estimates are semi-supervised.

The **spa** package is written for the R programming environment (R Development Core Team 2011) and available from the Comprehensive R Archive Network at `http://CRAN.R-project.org/package=spa`. This package provides the implementation of the sequential predictions algorithm. The package design consists of three general phases: (i) graph construction, (ii) constructing the spa object and (iii) post fitting functionality. The graph construction segment provides a series of functions to process observed graphs or to construct graphs from part of the feature data. Upon completing this stage an $n \times n$ adjacency matrix is obtained with diagonal elements representing nodes (observations in the data) and off-diagonal elements representing edge associations between nodes. For the second phase a SPA object is created from the inputs using semi-supervised smoothers (discussed in Section 3). One technical challenge common with smoothers is tuning parameter estimation. The SPA features sophisticated algorithms for data driven estimation of the smoother parameter based on a semi-supervised generalized cross-validation measure. Step (ii) completes with a SPA object that encapsulates ϕ.

The post fitting step consists of generic function utilities for manipulating and accessing a SPA object, such as `coef` and `fitted`, functions to `plot` and `print` the object, and functions for performing prediction/updating with the SPA object. The prediction/updating functionality for the package was carefully designed to distinguish between *transductive* and *inductive* prediction. Transductive prediction requires re-training or updating ϕ to use local proximity information with the intent of improving performance, or to incorporate a new set of observations that became available after training. This is most useful when the unlabeled data provide intrinsic structural information that can improve the general analysis with the classifier. For example, prediction with an observed graph, G, is inherently transductive since unlabeled data influence the topology of the graph (Culp and Michailidis 2008a); or in cases when the Z information provides a manifold in the data that is pertinent for classification of Y_L (Chapelle, Schölkopf, and Zien 2006). An inductive learner is designed to predict a new observation using the classifier without retraining. In this case the new observation does not provide any structural information and we are interested in classifying this case to the rule established. For example, constructing a contour plot of a classification rule is based off of predicting a grid determined by the range of the data. The grid has no inherent meaning and therefore transductive prediction is inappropriate.

The SPA classifier is both transductive and inductive, i.e,. transductive prediction is used to obtain the border which best represents the structure in the data, and then inductive prediction is used to classify new observations with respect to the border. The **spa** distinguishes between transductive and inductive prediction using the `update` and `predict` generics, respectively.

The author is aware of two software packages available for semi-supervised graph-based learning. The first is the **SGT-light** software which performs a SVM like approach on a graph to classify a response (Joachims 2003). The other is the Manifold Regularization software which can be adapted to fit a function to an observed graph (Belkin *et al.* 2004). Neither approach

to date performs internal parameter estimation, semi-parametric estimation, or is integrated in the R language. Indeed, the **spa** provides an R package that fits a semi-parametric graph-based estimator, that performs internal tuning parameter estimation, and that accommodates both transductive and inductive prediction.

In Section 2, we provide a simple example to gain insight for the graph-based classifiers implemented in this package. In Section 3, we provide analysis using **spa** in classification of the Cora text mining data set. In Section 4 we provide the methodology underpinning this approach, which motivates several key features of this approach including parameter estimation and transductive prediction. In Section 5, we provide the detailed layout of the **spa** package, discussing several functions offered for this package in graph construction (Section 5.1), object generation (Section 5.2), and post fitting (Section 5.3). In Section 6 we present two data examples, one involving classification on a manifold (Section 6.1) and the other involving protein interaction (Section 6.2). We close with some concluding remarks in Section 7.

2. Local classification on a graph

At the core of the proposed package are two intuitive algorithms designed to process a local similarity graph: (i) fitting the fits and (ii) the sequential predictions algorithm. To gain deeper insight into these approaches we first present the algorithms in an intuitive special case involving local similarity on two-dimensional planar graph. In Section 4, we rigorously describe the algorithmic details, assumptions and generalizations of these algorithms as implemented in the package.

In graph-based classification, the observations are vertices on a graph and the edges between observations describes local similarity. For example, in Figure 1 we present a simple lattice, where each observation is defined by the intersection of two grid lines. In the figure there are two labeled vertices, one is black and the other is white, while there are 34 unlabeled vertices. For classification the goal is to estimate the probability of each label belonging to either class, i.e., estimate probability class estimates (PCEs) for each vertex.

For fitting the fits, we initialize all unlabeled cases with PCEs of 0.5 (gray in the figure). Then we classify every observation as the weighted local average using this initialization for unlabeled cases and the true labels for labeled cases. The new probabilities for the unlabeled cases provide estimates for the unlabeled cases and are likely more precise than the initializations. Therefore, we now retrain the approach using the updated probabilities. This process is repeated until the unlabeled PCEs used in classification are the same as the unlabeled PCEs resulting from classification. This process is shown visually for the lattice example in Figure 1. The FTF algorithm is quite similar to semi-supervised harmonic approaches including random walks and label propagation (Chapelle *et al.* 2006, Chapter 11) but differs in that probabilities class estimates are also obtained for the labeled cases. The use of probability class estimates in the iterative phase of FTF for the unlabeled responses is referred to as *soft labels*, since they are not a hard binary responses. The latter is referred to as *hard labels* (Culp and Michailidis 2008a).

In the case of fitting the fits all unlabeled observations are treated equally in iterative training. The sequential predictions algorithm provides a way to correct for the use of unlabeled probabilities in training by penalizing vertices farther away from the labeled cases. The process is to first train observations one hop away from labeled cases, then penalize the PCEs resulting

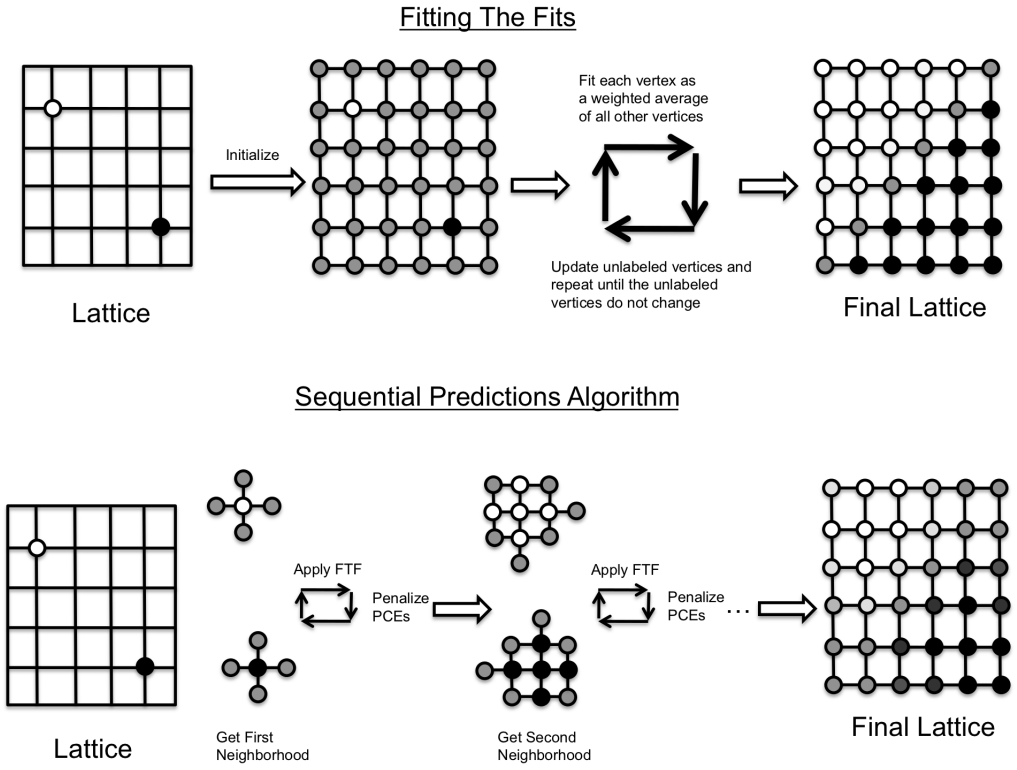

Figure 1: Process of classification on a 2-dimensional 36 node lattice using the (top) Fitting the fits algorithm and (bottom) sequential predictions algorithm. For this graph 2 vertices are labeled with the remaining 34 unlabeled.

from training towards the prior for Y_L. The process is repeated until all observations are classified. Refer to Figure 1, where here the final probabilities are less certain near the border compared to FTF.

3. Motivating text analysis application

In this section, we provide a prototype analysis with the **spa** package to illustrate the key functionality of this package on the Cora text data set (McCallum, Nigam, Rennie, and Seymore 2000). For these data, the observations are $n = 827$ text documents published predominantly in engineering, computer science and statistics journals. The attributes were generated electronically on the web and then processed with in-house scripts. The X information is 827×9 orthogonal matrix where $X_{ik} = I\{\text{paper } i \text{ is in journal } k\}$, for $k = 1, \cdots, 9$ corresponding to the 8 journals listed in Table 1 plus a category for all other journals. The G information is a weighted graph represented by a co-citation network between the text documents. Specifically, the adjacency matrix is constructed with $A_{ij} = \sum I\{i \text{ and } j \text{ cite the same document}\}$. The response is the binary indication that the specific document is classified to be about

Journal	AI	ML	Total
Artificial Intelligence	36	17	53
Journal of Artificial Intelligence Research	24	24	48
Machine Learning	2	32	34
IEEE Transaction on Pattern Analysis and Machine Intelligence	21	5	26
Neural Computing	3	20	23
IEEE Transactions on Neural Networks	1	16	17
Artificial Intelligence Magazine	11	1	12
Journal of the American Statistical Association	3	9	12
Other	340	258	598

Table 1: Journal frequencies for the classes under consideration in the Cora text data.

machine learning (ML) or artificial intelligence (AI). The data are available as part of the **spa** package.

To begin analysis, we provide the necessary preprocessing:

```
R> data("coraAI")
R> y <- coraAI$class
R> x <- coraAI$journals
R> g <- coraAI$cite
R> keep <- which(as.vector(apply(g, 1, sum) > 1))
R> setdiff(1:length(y), keep)

 [1]  12  19  48  49  50  52  68  90 103 129 151 174 178 193 212 255 259 284
[19] 293 296 297 325 327 362 370 458 459 461 465 490 497 507 546 567 577 593
[37] 606 612 691 720 724 725 726 733 734 742 805 819

R> y <- y[keep]
R> x <- x[keep,]
R> g <- g[keep, keep]
R> set.seed(100)
R> n <- dim(x)[1]
R> Ns <- as.vector(apply(x, 2, sum))
R> Ls <- sapply(1:length(Ns), function(i) sample(which(x[,i] == 1),
R+ ceiling(0.035 * Ns[i])))
R> L <- NULL
R> for(i in 1:length(Ns)) L <- c(L, Ls[[i]])
R> U <- setdiff(1:n, L)
R> ord <- c(L, U)
R> m <- length(L)
R> y1 <- y
R> y1[U] <- NA
```

Observations 12, 19, $\cdots$, 819 were removed since they were isolated vertices. A stratified sample is used to insure that every journal is represented in labeled and unlabeled partitions.

The spa is used to fit an estimate on the observed graph directly without the journal indication matrix (i.e., $\hat{Y} = \hat{f}(G)$):

```
R> A1 <- as.matrix(g)
R> gc <- spa(y1, graph = A1, control = spa.control(dissimilar = FALSE))
R> gc

Call:
spa(y1, graph = A1, control = spa.control(dissimilar = FALSE))

Sequential Predictions Algorithm (SPA) with training= 4 %

Type= soft  Parameter= 1.968709  GCV type:  approximate transductive

Fit Measures:

RMSE= 2.439   tGCV= 8.646   tDF=  5.623

Transductive Parameters:

Regions:  0   Regularization: Inf

Unlabeled Data Measures:

Training:   0.105 <=1 (Labeled/Supervised ~1)
Unlabeled:  0.965 <=1 (No Supervised Equivalent)
```

The output provides several descriptive aspects of the model fit. First notice that, the fit used 4% of the data as labeled during training. The fit measures provide standard fit description with root mean squared error, parameter estimation type (refer to Section 4) and transductive degrees of freedom. The transductive parameters were set to default, which means that the entire unlabeled data set was fit as one unit and no regularization was employed (i.e., FTF as a special case of SPA was employed). The unlabeled data measures provide a rough assessment of the potential usefulness of unlabeled data during training. The dissimilar flag must be FALSE for the spa.control to indicate that the co-citation network is a similarity matrix.

The natural concern is in the performance of the SPA on U (unlabeled set). To check performance on U, we calculate the misclassification error rate:

```
R> tab <- table(fitted(gc)[U] > 0.5, y[U])
R> 1 - sum(diag(tab)) / sum(tab)

[1] 0.4647355
```

The unlabeled error for this fit is extremely poor. An error this large typically indicates that the procedure is not using the unlabeled data appropriately during training. To investigate this, we provide a useful diagnostic by considering the A_{UL} partition of A. This partition provides the common links between labeled-unlabeled data, and if an unlabeled observation

has no such links (i.e., $A_{UL_i} \times \mathbf{1}_L = 0$) then the observation cannot be utilized directly for training. To assess the problematic unlabeled observations simply execute:

```
R> sum(apply(A1[U, L], 1, sum) == 0) / (n - m) * 100

[1] 39.92443
```

This tells us that nearly 40% of the cases have no direct associations with Y_L, which significantly deteriorates performance.

Next, we incorporate transductive prediction to account for unlabeled response estimates used in training by penalizing observations farther away from labeled cases. The idea is to sort the $A_{UL} \times \mathbf{1}_L$ information and break the data into regions enumerated from highest (most associated) to lowest (least associated). In other words, the vector $A_{UL} \times \mathbf{1}_L$ provides a sorting method that groups objects to be classified at subsequent rounds similar to the shortest path used in the lattice example in Section 2. The observations are then sequentially estimated, where at each round new unlabeled node estimates are treated as true values (refer to Section 4.3 for the general SPA or Section 2 for a simple example). To perform transductive prediction with the **spa** package use the `update` generic with the `dat` argument:[1]

```
R> pred <- update(gc, ynew = y1, gnew = A1, dat = list(k = length(U),
+    l = Inf))
R> tab <- table(pred[U] > 0.5, y[U])
R> 1 - sum(diag(tab)) / sum(tab)

[1] 0.2884131
```

The performance improves significantly. To update the `spa` object so that it corresponds to the above output execute:

```
R> gc <- update(gc, ynew = y1, gnew = A1,
+    dat = list(k = length(U), l = Inf), trans.update = TRUE)
R> gc

Sequential Predictions Algorithm (SPA) with training= 4 %

Transductive Parameters:

Regions:  794   Regularization: Inf

Unlabeled Data Measures:

Training:   0.105 <=1 (Labeled/Supervised ~1)
Unlabeled:  0.965 <=1 (No Supervised Equivalent)
```

[1]The FTF algorithm is the special case of SPA where all observations are in one group and there is no regularization, that is `dat = list(k = 0, l = Inf)` produces the FTF output.

We see that, the transductive parameters are updated with the new fit. The training unlabeled measure of 0.105 does not change when using transductive prediction.

Next, we investigate the estimate $\hat{Y} = X\hat{\beta} + \hat{f}(G)$ where X is the journal indication matrix and G is the co-citation network. To execute this with `spa`:

```
R> gjc <- spa(y1, x, g, control = spa.control(diss = FALSE))
R> gjc

Sequential Predictions Algorithm (SPA) with training= 4 %

Coefficients:
                             other            artificial_intelligence
                       0.369607329                        0.049032222
                  machine_learning                  neural_computing
                       0.959096925                        0.979303548
                   ieee_trans_Nnet                         ieee_tpami
                       0.947685447                        0.012443001
j_artificial_intelligence_research                        ai_magazine
                       0.006865285                        0.009279124
                              JASA
                       0.976590026

Transductive Parameters:

Regions:  0   Regularization: Inf

R> tab <- table((fitted(gjc) > 0.5)[U], y[U])
R> 1 - sum(diag(tab)) / sum(tab)

[1] 0.395466
```

The coefficient estimate is provided in the output and can be extracted using the `coef` generic. The output also shows that the estimate suffers from the same performance problems with respect to the unlabeled data as the previous fit. As before, this problem can be fixed by using transductive prediction:

```
R> gjc1 <- update(gjc, ynew = y1, xnew = x, gnew = A1,
+    dat = list(k = length(U), l = Inf), trans.update = TRUE)
R> gjc1

Sequential Predictions Algorithm (SPA) with training= 4 %

Type= soft  Parameter= 1.968709  GCV type:  approximate transductive

Coefficients:
                             other            artificial_intelligence
```

```
                        0.46182225                         0.34495785
                  machine_learning                 neural_computing
                        0.86100990                         0.91947929
                   ieee_trans_Nnet                          ieee_tpami
                        0.85709753                         0.20706484
j_artificial_intelligence_research                        ai_magazine
                       -0.05256496                         0.11758127
                              JASA
                        0.62853374

Transductive Parameters:

Regions:  794   Regularization: Inf

R> tab <- table((fitted(gjc1) > 0.5)[U], y[U])
R> 1 - sum(diag(tab)) / sum(tab)

[1] 0.197733
```

As we can see the performance of the approach improved significantly after we updated the model to include transductive prediction. The strong performance on the unlabeled data suggests that using both the X information and G information contributed to predicting the class label of a document. In addition, a coefficient vector is obtained that may provide interpretable value for assessing a specific journal's contribution to this classification problem.

4. Methodology

In Section 3, we provided an analysis illustrating several features utilized by the **spa** package. The output offers utilities useful for assessing and improving the fit, including transductive generalized cross validation and degrees of freedoms, influence of observations in U on $\hat{Y}_L$, an unlabeled contribution $\sum_i I\{A_{UL_i} \times \mathbf{1}_L = 0\}$ and a transductive prediction algorithm for processing data. Next we present the details of the algorithmic fitting methodology, which will also provide some justification for these measures of fit.

The proposed approach essentially extends the concept of a linear smoother into the semi-supervised setting using the fitting the fits algorithm (the sequential predictions algorithm is a generalization for which the package is named and presented in Section 4.3) (Culp and Michailidis 2008b). It is also quite similar to self-training approaches that use the expectation-maximization (EM) algorithm (Chapelle *et al.* 2006; Culp and Michailidis 2008b). The first step is to initialize the unlabeled vector $\hat{Y}_U^{(0)} = \vec{0}$. Then the FTF iteratively fits:

$$\begin{pmatrix} \hat{Y}_L^{(k+1)} \\ \hat{Y}_U^{(k+1)} \end{pmatrix} = \begin{pmatrix} S_{LL} & S_{LU} \\ S_{UL} & S_{UU} \end{pmatrix} \begin{pmatrix} Y_L \\ \hat{Y}_U^{(k)} \end{pmatrix}, \tag{1}$$

where S is a linear smoother for response $\hat{Y}$ and is independent of k. Notice that the labeled response estimate is reset to Y_L before fitting, which intuitively allows the procedure to train with the observed response (known response) and use the previous estimated response fit

where the responses are unobserved (unknown). The algorithm is repeatedly executed until global convergence defined by $\| \hat{Y}_U^{(k+1)} - Y_U^{(k)} \| < \epsilon$ with $\epsilon > 0$. The vector $\hat{Y} = [\hat{Y}_L^\top, \hat{Y}_U^\top]^\top$ denotes the convergent solution to (1). To establish global convergence, notice that at step k we have that $\hat{Y}^{(k)} = \sum_{\ell=0}^{k-1} S_{UU}^\ell S_{UL} Y_L + S_{UU}^k \hat{Y}_U^{(0)}$ and therefore the geometric matrix series is guaranteed to converge whenever $\rho(S_{UU}) < 1$ where ρ is the spectral radius. The closed form convergent solution can be obtained as the limit on the geometric matrix series: $\hat{Y}_L = S_{LL} Y_L + S_{LU} \hat{Y}_U$ and $\hat{Y}_U = (I - S_{UU})^{-1} S_{UL} Y_L$.

The **spa** package primarily fits stochastic based smoothers including kernel smoothers, graph smoothers and a semi-parametric extension involving kernel smoothing. However, before we study kernel smoothers in the semi-supervised setting, we first look at linear regression to gain important insight into the procedure, since several key aspects of the **spa** package are motivated from linear regression. Upon presenting FTF with linear regression (Section 4.1), we then focus on kernel smoothing and graph smoothing (Section 4.2), the full sequential predictions algorithm (Section 4.3), and semi-parametric smoothing (Section 4.4) in the semi-supervised setting.

4.1. Example: Fitting the fits with linear regression

Next we study linear regression in the semi-supervised setting to provide insight and motivate key results, including transductive degrees of freedom and transductive generalized cross-validation. To implement semi-supervised linear regression, one must apply the hat matrix

$$H = X(X^\top X)X^\top = \begin{pmatrix} X_L(X^\top X)^{-1}X_L^\top & X_L(X^\top X)^{-1}X_U^\top \\ X_U(X^\top X)^{-1}X_L^\top & X_U(X^\top X)^{-1}X_U^\top \end{pmatrix} = \begin{pmatrix} H_{LL} & H_{LU} \\ H_{UL} & H_{UU} \end{pmatrix}$$

as the smoother in (1). Note the supervised hat matrix is $\widetilde{H}_{LL} = X_L(X_L^\top X_L)^{-1}X_L^\top$. Semi-Supervised linear regression with FTF is quite simple to fit in R, using the `.GlobalEnv` nested in `sapply` command:

```
R> set.seed(100)
R> x <- matrix(rnorm(40), 20)
R> z <- cbind(x, 1)
R> y <- z %*% rnorm(3) + rnorm(20, , 0.5)
R> H <- z %*% solve(t(z) %*% z, t(z))
R> L <- 1:5
R> U <- 6:20
R> yh <- c(y[L], rep(0, 15))
R> ftfls <- function(i) {
+    .GlobalEnv$yh[L] <- y[L]
+    .GlobalEnv$yh <- H %*% .GlobalEnv$yh
+  }
R> ftfres <- sapply(1:200, ftfls)

[1] -1.8053630 -0.7672166  ... -0.9983772
```

Upon convergence an estimate for the response is obtained in closed form with $\hat{Y}_U = (I - H_{UU})^{-1} H_{UL} Y_L$ and $\hat{Y}_L = H_{LL} Y_L + H_{LU} \hat{Y}_U$.

The natural question to ask is, "What are the properties of the convergent solution for linear regression with FTF?" The answer is clarified by comparing our result to the ordinary supervised least squares solution:

```
R> dat <- data.frame(y = y, x)
R> supls <- predict(lm(y ~ ., data = dat, subset = L), newdata = dat)
R> apply((ftfres - supls)^2, 2, sum)[c(1:2, 199:200)]

[1] 2.831134e+01 1.668253e+01 ...  8.545281e-15 7.265162e-15
```

By iteration 200 the difference between the two is negligible and therefore we see that semi-supervised linear regression using FTF reduces to the supervised least squares solution. This equivalency is proven for the generalized linear model in Culp and Michailidis (2008b). A similar result can be shown when treating the semi-supervised linear regression problem as a missing data problem (Little and Rubin 2002, p. 29)

The equivalency between semi-supervised linear regression using FTF and supervised ordinary least squares results in two new expression for the hat matrix. The equivalent hat matrices are given by:

$$\begin{aligned} \widetilde{H}_{LL} &= X_L(X_L^\top X_L)^{-1}X_L^\top \text{ (supervised)} && (2)\\ &= H_{LL} + H_{LU}(I - H_{UU})^{-1}H_{UL}\text{(transductive)} && (3)\\ &= H_{LL} + \widetilde{H}_{UL}^\top H_{UL} \text{ (approximate transductive)} && (4) \end{aligned}$$

where $\widetilde{H}_{UL} = X_U(X_L^\top X_L)^{-1}X_L^\top$ (supervised unlabeled estimate). Upon convergence each formulation has the same error and the same trace. The degrees of freedom of the residuals of the semi-supervised estimator are defined as

$$df = \mid L \mid -\text{tr}(H_{LL} + H_{LU}(I - H_{UU})^{-1}H_{UL}) = \mid L \mid -p$$

since this form yields the smoother for the labeled estimator resulting from FTF, i.e., $\hat{Y}_L = \left(H_{LL} + H_{LU}(I - H_{UU})^{-1}H_{UL}\right)Y_L$.

This equivalency between these projection matrices is useful in computing the generalized cross-validation (GCV) error rate of a smoother, i.e., the GCV for smoother S is defined as:

$$\text{GCV}(S) = \frac{\|Y_L - SY_L\|_2^2}{(1-\text{tr}(S)/|L|)^2}.$$

To do this, we define GCV with reference to (2) as labeled GCV (`"lGCV"`), GCV with reference to (3) as transductive GCV (`"tGCV"`), and GCV with reference to (4) as approximate transductive GCV (`"aGCV"`). All of these measures provide an identical value in linear regression but will differ in kernel smoothing (refer to Section 4.2). The `"tGCV"` criterion would be the best choice but is computationally demanding due to the need for $(I - H_{UU})^{-1}$. We recommend using the approximate transductive GCV criterion since it is less demanding computationally and is still influenced by observations in U. The choice of GCV is set by `spa.control` with `"aGCV"` as the default.

Remark: Linear regression is not implemented in the **spa** package since the `lm` command computes it equivalently.

4.2. Semi-supervised stochastic kernel smoothing

Next we study kernel smoothers, where a kernel function is applied directly to distances in X with $W_{ij} = K_\lambda(x_i, x_j)$ where $K_\lambda(x_i, x_j) = \exp(-d_{ij}/\lambda)$ is default. In general λ is a tuning parameter where $\lambda \approx 0$ tends to yield solutions with low bias and high variance while $\lambda \to \infty$ tends to yield solutions with high bias and low variance. The matrix W emits the partition for $i, j \in L \cup U$:

$$W = \begin{pmatrix} W_{LL} & W_{LU} \\ W_{UL} & W_{UU} \end{pmatrix}.$$

The FTF algorithm allows for the observations in W_{UU}, W_{UL}, and $W_{UL}^\top$ to influence training of the procedure, by iteratively fitting (i) $\hat{Y}_L^{(k-1)} = Y_L$ and (ii) $\hat{Y}^{(k)} = S\hat{Y}^{(k-1)}$ with $S = D^{-1}W$ where D is the diagonal row sum matrix of W (denoted as $D = \text{diag}(W\mathbf{1})$) and initialization $\hat{Y}_U^{(0)}$. Tracing the iteration, we get that the unlabeled estimator is $\hat{Y}_U^k = \sum_{\ell=0}^{k-1} S_{UU}^\ell S_{UL} Y_L + S_{UU}^k \hat{Y}_U^{(0)}$ and take the limit as $k \to \infty$ to get the closed form solution, $\hat{Y} = \hat{Y}^{(\infty)}$:

$$\begin{pmatrix} \hat{Y}_L \\ \hat{Y}_U \end{pmatrix} = \begin{pmatrix} S_{LL} + S_{LU}(I - S_{UU})^{-1} S_{UL} \\ (I - S_{UU})^{-1} S_{UL} \end{pmatrix} Y_L. \tag{5}$$

The condition for producing a stable estimator is discussed in Section 4.3.

Unlike in linear regression, $\hat{Y}_L$ is not equivalent to the supervised estimator. However, the supervised estimator is influential on the result as we show next. First for kernel regression we have that $S = D^{-1}W$. Define D_L, D_U as the respective labeled/unlabeled diagonal submatrices of the diagonal row-sum matrix D. In addition, denote $D_{LL} = \text{diag}(W_{LL}\mathbf{1}_L)$, that is the row sum matrix of W_{LL}, and similarly denote $D_{LU} = \text{diag}(W_{LU}\mathbf{1}_U)$, $D_{LU} = \text{diag}(W_{LU}\mathbf{1}_U)$, and $D_{UU} = \text{diag}(W_{UU}\mathbf{1}_U)$. From this, $D_L = D_{LL} + D_{LU}$ and $D_U = D_{UU} + D_{UL}$. Next, we denote the supervised kernel smoother as $T_{LL} = D_{LL}^{-1} W_{LL}$ and the supervised prediction smother as $T_{UL} = D_{UL}^{-1} W_{UL}$. The supervised estimator is denote by $\widetilde{Y}$. To connect supervised estimation to $\hat{Y}_L$ we have that:

$$\begin{aligned} \hat{Y}_L &= \left(D_L^{-1} D_{LL}\right) T_{LL} Y_L + \left(I - D_L^{-1} D_{LL}\right) S_{LUL} Y_L \\ &\equiv Q\widetilde{Y}_L + (I - Q)\breve{Y}_L, \end{aligned} \tag{6}$$

where $Q = D_L^{-1} D_{LL} \leq I$ and $S_{LUL} = D_{LU}^{-1} W_{LU}(I - S_{UU})^{-1} S_{UL}$. From this decomposition, we notice that if $Q \approx I$ then the supervised labeled estimate is close to the semi-supervised labeled estimate. On the other hand, shrinking this matrix provides more influence with S_{LUL}. It is easy to show that S_{LUL} is right stochastic and therefore (6) is an average of two right stochastic matrices which is also right stochastic. The value $\bar{Q} \equiv \frac{\text{tr}(Q)}{|L|}$ provides a quick determination of the effect of the unlabeled data on the labeled estimate in training and is provided in the output of the **spa** object (0.105 for the Cora data). Values near one indicated that (6) is close to $\widetilde{Y}_L = T_{LL} Y_L$ (supervised) while smaller values indicate that the estimate of $\hat{Y}_L$ is influenced by $\breve{Y}_L$.

For the kernel smoothing problem, we are also interested in approximating the matrix S_{LUL} without requiring the computation of the inverse (for λ estimation). Using the linear regression result, we employ $S_{LUL} \approx V^{-1} T_{UL}^\top S_{UL} = V^{-1} T_{UL}^\top (I - D_U^{-1} D_{UU}) T_{UL}$, where V is the appropriate matrix to make the approximation right stochastic. This adjustment is extremely useful for fitting GCV as it is used by `type = "aGCV"` and is default for `spa.control`.

This result in the case of kernel regression is not equivalent to GCV using the transductive smoother or GCV using the labeled smoother (refer to Section 4.1).

Graph-based smoothing is a natural generalization of kernel smoothing. The adjacency matrix of the graph is given by A and the smoother is obtained as $S = D^{-1}A$. The FTF is then applicable in the same manner as with the kernel smoothing approach discussed previously (i.e., A and W are quite similar). One key difference between graph-based smoothing and kernel smoothing is that the graph problem is inherently semi-supervised (often referred to as transductive in this special case), and therefore a supervised alternative does not exists. Care is taken with the design of the **spa** to allow for transducitve prediction when necessary.

4.3. The sequential predictions algorithm

The FTF algorithm was previously employed with smoothers. The full sequential predictions algorithm is an extension of FTF to perform transductive prediction with the intent of improving performance (refer to Cora example). Transductive prediction addresses the issue of updating a classification or regression technique to incorporate either new observations (labeled or unlabeled) unavailable originally during training, reassess the current unlabeled observations used previously during training, or both.

To motivate the need for the SPA algorithm, we first consider the circumstances for when $(I - S_{UU})^{-1}$ is expected to be unstable. To provide insight recall that the smoother $S = D^{-1}W$ is right stochastic, i.e.,

$$\begin{pmatrix} S_{LL} & S_{LU} \\ S_{UL} & S_{UU} \end{pmatrix} \begin{pmatrix} \mathbf{1}_L \\ \mathbf{1}_U \end{pmatrix} = \begin{pmatrix} \mathbf{1}_L \\ \mathbf{1}_U \end{pmatrix} \text{ and } S_{ij} \geq 0.$$

In such a circumstance, a stable estimator is achieved whenever $S_{UU}\mathbf{1}_U < \mathbf{1}_U$ (i.e., the sum of each row in S_{UU} is less than one). From the above equation we observe that the requirement is satisfied whenever $S_{UL}\mathbf{1}_L > \mathbf{0}_L$ (i.e., every row in S_{UL} sums to a positive number). The instability occurs whenever $S_{UL}\mathbf{1}_L \approx \mathbf{0}_L$ or sufficiently $W_{UL} \times \mathbf{1}_L \approx \mathbf{0}_L$. This condition can easily be violated in practice. For example, a compact kernel could force weak connections between labeled and unlabeled data to vanish. It is also common to employ a non-compact kernel and then a k-nn function to that kernel (known as a k-nn graph), which results in the analogous effect. In the case of an observed graph, it is not difficult to obtain a vertex that is only connected to other unlabeled vertices (refer to the Cora text data). This problem becomes worse as $\mid L \mid$ decreases for each of these examples, since there are less labeled cases to connect with the unlabeled cases.

To correct for this we extend the FTF algorithm into the sequential predictions algorithm (Culp and Michailidis 2008a). The main idea is to sort $W_{UL} \times \mathbf{1}_L$ and partition the unlabeled data $U = \{U^{(\ell)}\}_{\ell=1}^{k}$ into k disjoint sets such that the set with the most labeled connectivity is employed first with FTF. In other words, apply FTF with labeled cases $L^{(0)} = L$ and unlabeled cases in $U^{(1)}$. Then treat the unlabeled estimates as true and at iteration ℓ train FTF with labeled cases $L^{(\ell)} = L^{(\ell-1)} \cup U^{(\ell)}$ and unlabeled set $U^{(\ell+1)}$. This process repeats with a total of k calls to FTF. In Culp and Michailidis (2008a), we provide a penalty function based on constrained regularization to correct for the use of unlabeled predictions at subsequent rounds. The main idea is to increasingly push the estimates towards a global point estimates (mean of Y_L) as the algorithm progresses. This entire approach is implemented in the `update` generic, requiring parameter `dat = list(k = 0, l = Inf)` by default with subparameters `k`

(number of regions) and `l` (inverse regularization penalty, i.e., `l = 0` is infinite regularization while `l = Inf` is no regularization). The approach is shown to work quite well for the Cora text data and was also shown to be competitive to several state-of-the-art graph-based semi-supervised approaches. An extension to hard labels based on exponential loss is also discussed in Culp and Michailidis (2008a) and is implemented in **spa** (use `type = "hard"` for `spa` call and `reg = "hlasso"` for `update`).

4.4. An optimization framework for the semi-parametric model

The fitting the fits algorithm provides a simple and effective way to fit data with an unlabeled partition and or graph based terms. The connection to optimization is now established for fitting semi-parametric models. In order to do this, we first establish the link between the FTF algorithm and optimization. Then we generalize this approach to the more general semi-parametric problem.

For this, denote $\phi(y, f) = (y-f)^2$ as the squared error loss function, and consider the following optimization problem:

$$\min_f \sum_{i \in L} \phi(y_i, f_i) + \lambda f^\top P f,$$

where P is a positive semi-definite penalty matrix. In this construct the loss is incurred from the labeled estimator f_L to the observed response Y_L, while the full function is accounted for with the penalty term. The penalty matrix P is taken as the combinatorial Laplacian $P = \Delta \equiv D - W$ for observed graph and constructed graph terms.

Next we demonstrate that the FTF algorithm with kernel regression does indeed solve the following optimization problem:

$$\min_f (Y_L - f_L)^\top W_{LL}(Y_L - f_L) + f^\top \Delta f.$$

Define, $Y(Y_U) = [Y_L,\ Y_U]$ with Y_U an arbitrary unlabeled response. Then we have that $(Y_L - f_L)^\top W_{LL}(Y_L - f_L) = (Y(f_U) - f)^\top W(Y(f_U) - f)$ for all f and is independent of f_U. From this, the derivative solves $-W(Y(\hat{f}_U) - \hat{f}) + \Delta \hat{f} = \vec{0}$. Rearranging we get that:

$$\begin{pmatrix} \hat{f}_L \\ \hat{f}_U \end{pmatrix} = \begin{pmatrix} S_{LL} & S_{LU} \\ S_{UL} & S_{UU} \end{pmatrix} \begin{pmatrix} Y_L \\ \hat{f}_U \end{pmatrix},$$

where $S = (W + \Delta)^{-1} W = D^{-1} W$. Solving the fixed point $\hat{f}_U = S_{UU}\hat{f}_U + S_{UL} Y_L$ and plugging the solution back in yields the identical solution to (5) for kernel regression.

Using this approach, we then motivate the semi-parametric model $\eta = X\beta + f(G)$ as the solution to:

$$\min_{f,\beta} \sum_{i \in L} (Y_L - f_L - X_L\beta)^\top W_{LL}(Y_L - f_L - X_L\beta) + f^\top \Delta f.$$

As we proceeded in kernel regression, we have that $(Y(\eta_U) - \eta)^\top W(Y(\eta_U) - \eta) = (Y_L - \eta_L) W_{LL}(Y_L - \eta_L)$ and is independent of η_U. Taking the gradient with respect to f we get that

$$-W(Y(\eta_U) - \eta) + \Delta f = \vec{0} \implies f = S(Y(\eta_U) - X\beta) \text{ with } S = (W + \Delta)^{-1} W = D^{-1} W.$$

Taking the gradient with respect to β and substituting for f we get that:

$$-X^\top W(Y(\eta_U) - \eta) = \vec{0} \implies X^\top W(I - S)X\beta = X^\top W(I - S)Y(\eta_U).$$

From this, we get that $\hat{\eta} = X\hat{\beta} + \hat{f} = X\hat{\beta} + S(Y(\hat{\eta}_U) - X\hat{\beta}) = H(S)Y(\hat{\eta}_U)$ for some smoother $H(S)$. From this, the FTF for the semi-parametric estimator reduces to: initialize $\hat{\eta}_U^{(0)}$ and repeat, (i) $\hat{\eta}_L^{(k)} = Y_L$ and (ii) $\hat{\eta}^{(k+1)} = H(S)\hat{\eta}^{(k)}$. Convergence is guaranteed whenever $\rho(H(S)_{UU}) < 1$. The SPA algorithm naturally applies to the semi-parametric model where the estimates for β are built up over the k calls based on the order of $W_{UL} \times \mathbf{1}_L$.

5. The spa package

The **spa** package provides semi-supervised functions in R to fit and evaluate an estimate of the form $\hat{Y} = \hat{f}(G)$ or $\hat{Y} = X\hat{\beta} + \hat{f}(G)$, where $\hat{\beta}$ is the coefficient vector, $\hat{Y}$ is the estimated response, and $\hat{f}$ is a function defined over the vertices of G. The process is broken into three key phases: (i) processing the graph, (ii) object generation and (iii) post fitting (refer to Figure 2). The first phase provides a series of functions for processing an observed graph (refer to the Cora example above) or a graph constructed from the Z data. The second phase is generating the `spa` object with the tuning parameter estimation functionality. The third phase provides post fitting functions featuring transductive prediction with the `update` generic, inductive prediction with the `predict` generic, and a series of other standard generic functions.

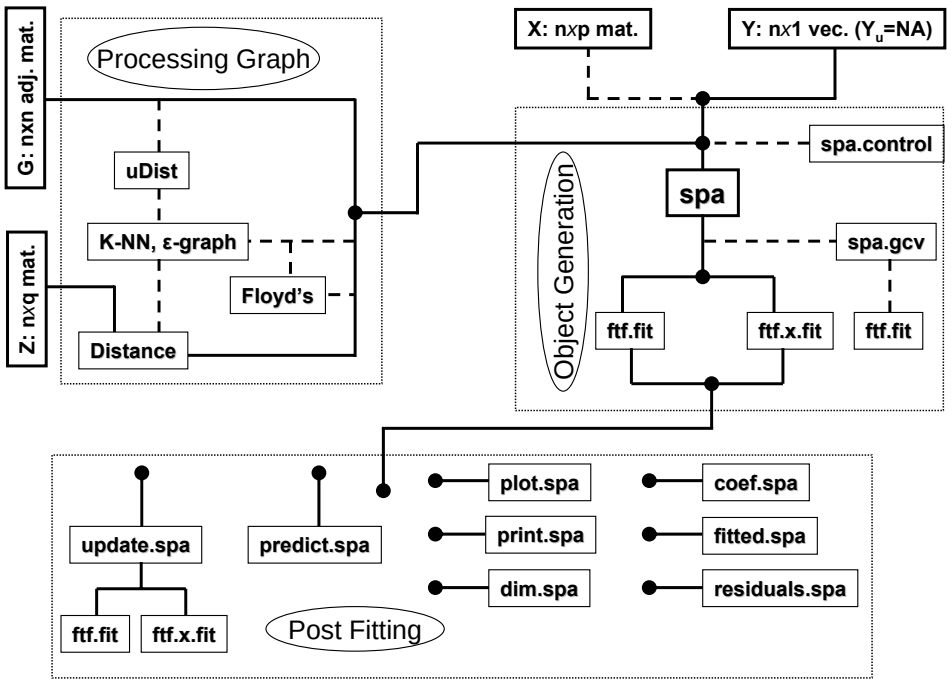

Figure 2: Sequence diagram for describing the relationship between the functions in the **spa** package. Dashed lines indicate steps that are optional.

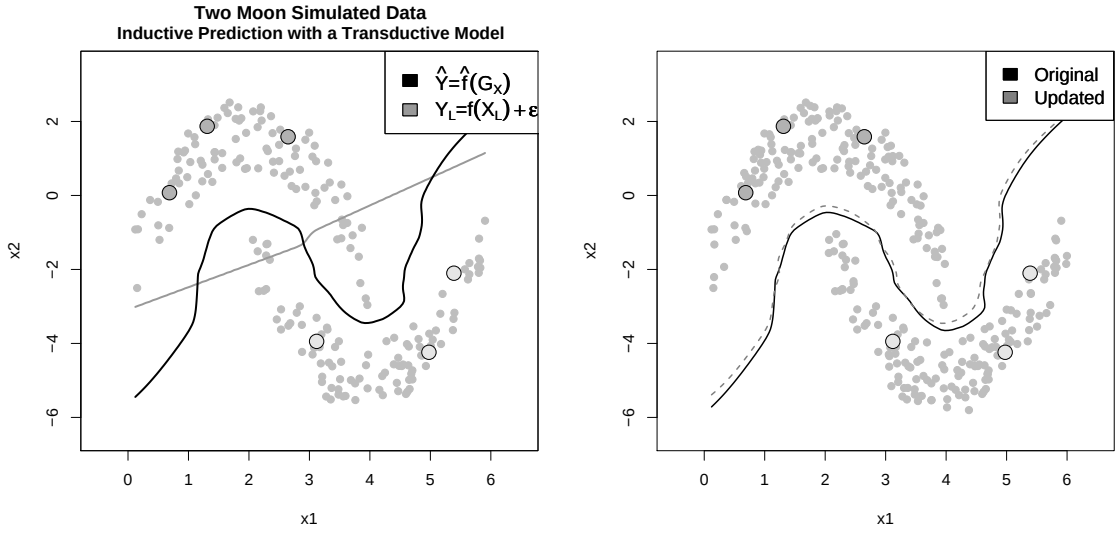

Figure 3: Scatterplot for two variables X_1 and X_2 with the class label as light gray, dark gray, or missing (small gray circle). (left) A supervised kernel smoother is gray, and the SPA fit is black. (right) Transductive prediction with the `spa`, the original fit for the two moon data set is provided as black, while the retrained `spa` was gray and dashed.

To illustrate the key functions necessary for fitting a model with the **spa** package we consider a synthetic two moon data set (refer to Figure 3). The unlabeled data are generated based on the intuitive *cluster assumption*: If the classes are linearly separable then observations on similar clusters should share the same label. In this example, the SPA breaks the unlabeled data into two moon-like clusters, a separation that cannot be found by the supervised kernel smoother.

To load the data in R execute:

```
R> library("spa")
R> set.seed(100)
R> dat <- spa.sim(type = "moon")
```

The three phases for **spa** presented in Figure 2 are each discussed in detail for these data.

5.1. Processing graph

The graph processing stage highlighted in Figure 2 provides the means necessary for preparing the Z information or an observed graph for processing with the `spa` function. The functions for manipulating these data consist of `uDist`, `knnGraph`, `epsGraph`, and `floyd`.

In the case of an observed graph, this step accounts for several diverse types of similarity graphs. For example, if the graph were sparse (e.g., a planar graph) then one may wish to employ the shortest path distance metric on the graph using the `uDist` function. The purpose of performing this manipulation with a sparse graph is to relate distant nodes with a distance proximity measure through neighboring links. As a consequence the unlabeled observations that were not originally directly linked to labeled cases may now have associations

through neighboring unlabeled observations that were connected with labeled cases. On the other hand, when the graph is dense, like with the Cora example, this type of manipulation is not necessary and in our experience can hinder performance. If the `uDist` function is employed with an observed graph, then the dissimilar flag in the `spa.control` must remain `TRUE` otherwise it must be set to `FALSE`.

In the case of converting Z information to a graph, the first step is to specify an appropriate distance metric on Z. For the simulated data, we would employ Euclidean distance:

```
R> Dij <- as.matrix(daisy(dat1[, -1]))
```

For Z matrices with large q, restricting to local connectivity with a k-nn or ϵ graph can significantly improve performance of this technique (Tenenbaum, Silva, and Langford 2000). However, the k-nn object is not a proper distance metric, and we often wish to convert into a proper metric with Floyd's algorithm (the triangle inequality does not hold for a k-nn graph). To do this with **spa** execute:

```
R> DNN <- knnGraph(Dij, dist = TRUE, k = 5)
R> DFL <- floyd(DNN)
```

The flag `dist = TRUE` indicates that the object is distance weighted.

Upon obtaining the appropriate graph representation for the `spa` function, we focus next on generating a `spa` object. Notice the other `spa` input parameters, Y and X, require minor processing. For Y, set $Y_U = NA$ and make sure X is either a `data.frame` or `matrix`.

5.2. Generating a `spa` object

The `spa` object encapsulates the model fit by the procedure. This step accounts for parameter estimation, problematic unlabeled data, and object generation. The parameter estimation procedure is designed to estimate parameters efficiently using one of four generalized cross-validation measures with either a bisection algorithm or grid search. The problematic unlabeled data cases ($W_{UL_i} \times \mathbf{1}_L = 0$) are carefully identified and handled with the `global` argument. The final object is then generated, reporting several useful measures that describe the current fit. The functions employed here include `spa.control`, `spa` and internal GCV/fitting functions.

For parameter estimation, one must first consider the `spa.control` function. For this, the specified GCV is optimized over a set of potential λ candidates, or a value is provided using the `ltval` option. To determine the set of potential candidates we provide a fast bisection algorithm and a slow, more thorough, grid search. For the bisection algorithm the key parameters are `lqmax = 0.2, lqmin = 0.05, ldepth = 10`, and `ltmin = 0.05`. The `lqmax` and `lqmin` set a range to search based on the [`lqmin`, `lqmax`] $\cdot 100$ quantile of the density taken on finite distances. In our experience, this provides a fairly good spread of potential λ candidates. In some cases, the quantile determined by `lqmin` is negative due to a large density at zero, and one can set the parameter `ltmin` to correct for this. The bisection algorithm recursively divides the interval in half and finds the subinterval that contains the minimum GCV. This process is repeated `ldepth` times. The approach is fast and usually works well. However the algorithm can get stuck and arrive at a λ estimate that is unsuitable. To correct for this, a uniform search over a grid of size `lgrid` between the `lqmin` to `lqmax` quantiles can be performed to estimate λ.

The object generation required specification of the unlabeled data. This can be done in two ways. First a labeled set of indices can be provided using the `L` argument, and the second approach is to set $Y_U = NA$. The algorithm processes a supervised kernel smoother if no unlabeled data are specified. To generate the `spa` object with the simulated moon example, we execute the following command in R:

```
R> L <- which(!is.na(dat$y))
R> U <- which(is.na(dat$y))
R> gsup <- spa(dat$y[L], graph = Dij[L, L])
R> gsemi <- spa(dat$y, graph = Dij)
```

First we consider the supervised kernel smoother fit on only the labeled data:

```
R> gsup

Sequential Predictions Algorithm (SPA) with training= 100 %

Type= soft  Parameter= 0.04219128  GCV type:  labeled

....

Unlabeled Data Measures:

Training:   1 <=1 (Labeled/Supervised ~1)
Unlabeled:  NaN <=1 (No Supervised Equivalent)
```

For the supervised kernel smoother, the parameter λ was estimated as 0.04. Notice also that if no unlabeled data are specified then the parameter estimation defaults to using labeled GCV. In addition, the unlabeled data measures have no meaning here. The supervised kernel smoother result is shown as the gray curve in both panels of Figure 3. Notice that this estimate completely misses the moon like structure in the left figure.

Next, we consider the semi-supervised `spa` fit on the moon data:

```
R> gsemi

Sequential Predictions Algorithm (SPA) with training= 3 %

Type= soft  Parameter= 0.05  GCV type:  approximate transductive

...

Unlabeled Data Measures:

Training:   0.848 <=1 (Labeled/Supervised ~1)
Unlabeled:  0.995 <=1 (No Supervised Equivalent)
```

The approximate GCV is used to estimate λ as 0.05. The first unlabeled data measure is $\frac{1}{|L|}\text{tr}(D_{LL}^{-1}D_L) = 0.848$. The semi-supervised result corresponds to the black curve in left

panel of Figure 2. Here the use of the unlabeled data produces a classification border that separates the moon-like data clusters.

To view the GCV path for parameter estimation consider the `parm.est` structure:

```
R> gsemi$model$parm.est
```

```
$cvlam
[1] 0.05
$gcv
 [1] 3.414852e-29 2.327041e-01 2.396981e-02 4.819773e-04 5.030481e-07
 [6] 2.823065e-11 3.441998e-16 9.467595e-21 1.006386e-20 6.494275e-20
$lambda
 [1] 0.05000000 1.27511510 0.66255755 0.35627877 0.20313939 0.12656969
 [7] 0.08828485 0.06914242 0.05957121 0.05478561
```

We see that the approach considered several values of λ before settling on $\lambda = 0.05$.

5.3. Post fitting

The **spa** package provides several enhancements for processing a `spa` object post fitting. Most are standard generic methods associated with models in general. The main focus is on transductive and inductive prediction. The generics available at this stage are: `update`, `predict`, `plot`, `print`, `dim`, `coef`, `fitted`, and `residuals`.

For inductive prediction, the `predict` generic is designed to process a new vertex, where the edges for this observation with respect to the original labeled and unlabeled observations are assumed to be available. That is, let $w_{\cdot j}$ be the edge network for predicting vertex $\cdot$, and form the prediction as $\hat{Y}_\cdot = \frac{\sum_{j \in L \cup U} w_{\cdot j} \hat{Y}_j}{\sum_{j \in L \cup U} w_{\cdot j}}$ with $\hat{Y} = [Y_L^\top, \hat{Y}_U^\top]^\top$. Next we illustrate inductive prediction for observation $x = (0\ 0)$ with `spa.predict`:

```
R> newobs <- c(0, 0)
R> newnode <- as.matrix(sqrt(apply(cbind(dat$x1 - newobs[1],
+    dat$x2 - newobs[2])^2, 1, sum)))
R> round(predict(gsemi, gnew = newnode), 3)
```

```
[1] 0
```

One could also process a sequence of new data points at once, e.g., to predict $(4, -4), \ldots, (4, 4), (4, 5)$ execute:

```
R> newnodes <- sqrt(t(sapply(1:10 - 5,
+    function(j) (dat$x1 - 4)^2 + (dat$x2 - j)^2)))
R> round(predict(gsemi, gnew = newnodes), 3)
```

```
 [1] 1.000 0.075 0.034 0.001 0.000 0.000 0.000 0.000 0.000 0.000
```

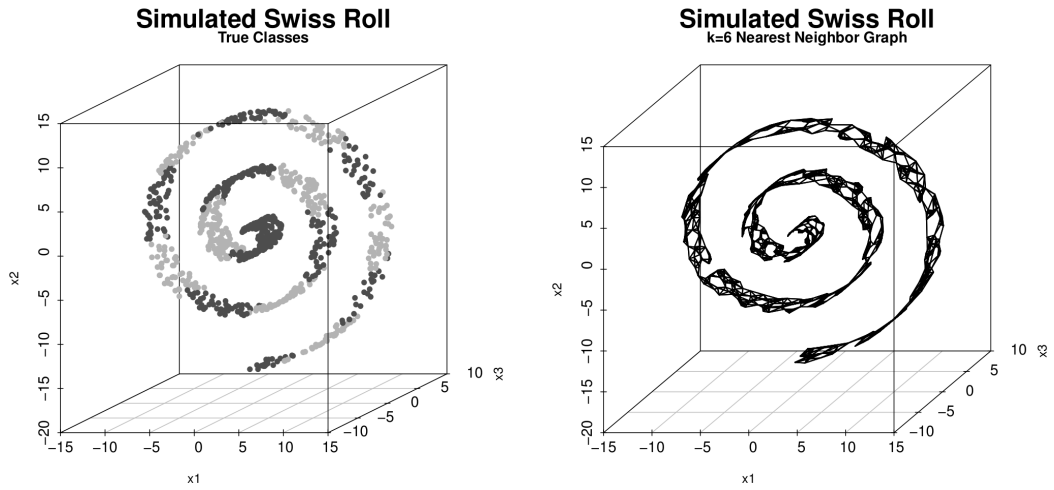

Figure 4: A synthetic Swiss Roll classification set. The true classes are provided on the left, and the $k = 6$ nearest neighbor graph is given on the right.

One could classify these observations according to the above probability class estimates.

Transductive prediction (updating) is more complex. In this case the observations provide inherent meaning and the current classifier should be modified to incorporate them. The object is changed post fitting, and therefore this type of prediction is really an update. The purpose of this `update` could be to perform the general SPA on the observations used for training with different settings or to perform the general SPA with new observations (labeled or unlabeled). The key parameters other than the (y, x, G) data for retraining include `dat` and `trans.update`. The `dat` parameter is a list with two components: $k = 0$ and $l =$Inf (default). The parameter k controls the number of regions to break U into, while l controls the regularization for penalizing estimates based on its regions (regions farther from the original L get higher regularization). To update with a new unlabeled data set generated from the same underlying data mechanism, execute:

```
R> dat <- rbind(dat, spa.sim(100, 0))
R> gsemi <- update(gsemi, ynew = dat$y, , as.matrix(daisy(dat[, -1])),
+    trans.update = TRUE)
```

The R `contour` plot for the new updated border is given in Figure 3 (right).

6. Examples

6.1. Swiss roll data set

For the first example, we present a simulation where the variables in the X data lie on a swiss roll manifold (refer to Figure 4, left). For the response, we first define a sequence of regions denoted by index i then the classification rule is defined as the indicator that an observation is on an even region. Refer to Figure 4 (left) for the region breakdown. The technical details

for the example are as follows: let $z_1 \sim U[0, 5 * \pi]$ and $z_2 \sim U[6, 10]$ then we have that the manifold is generated with $x_1 = z_1 \sin(z_1)$, $x_2 = z_1 \cos(z_1)$, and $x_3 = z_2 - 8$. For classification, set $x^2 = \sum_{i=1}^{3} x_i^2$ and then generate the response with:

$$y[(x^2 < 15(i+1) \;\; \& \;\; x^2 \geq 15i)] = (i \bmod 2) \tag{7}$$

where $i = 0$ to $\lceil \frac{\max_i x_i^2}{15} \rceil$.

The R code necessary to generate the example is given below:

```
R> library("scatterplot3d")
R> n <- 1000
R> set.seed(100)
R> z1 <- runif(n, 0, 5 * pi)
R> z2 <- runif(n, 6, 10)
R> x1 <- z1 * sin(z1)
R> x2 <- z1 * cos(z1)
R> x3 <- z2 - 8
R> x <- cbind(x1, x3, x2)
R> xsq <- apply(x^2, 1, sum)
R> y <- rep(100, n)
R> inc <- 15
R> p1 <- ceiling(max(xsq)/inc)
R> for(i in 0:p1) y[xsq < inc * (i + 1) & xsq >= i * inc] = i %% 2
R> gr <- gray(c(0.3, 0.7))[y + 1]
R> scatterplot3d(x1, x3, x2, ylim = c(-8, 8), color = gr, pch = 16,
+    cex.symbols = 1, cex.axis = 1.2, angle = 30)
R> title("Simulated Swiss Roll", cex.main = 2)
R> title("\n\nTrue Classes")
R> g1 <- knnGraph(x, k = 6, weighted = FALSE)
R> pl <- scatterplot3d(x1, x3, x2, ylim = c(-8, 8), color = gr, pch = 16,
+    cex.symbols = 1.2, type = "n", cex.axis = 1.2, angle = 45)
R> for(i in 1:n) {
+    ind <- which(g1[, i] > 0)
+    for(j in 1:length(ind)) {
+      pnts <- pl$xyz.convert(c(x[i, 1], x[ind[j], 1]),
+        c(x[i, 2], x[ind[j], 2]), c(x[i, 3], x[ind[j], 3]))
+      lines(pnts$x,pnts$y, col = 1, lwd = 1.5, lty = 1)
+    }
+  }
R> title("Simulated Swiss Roll", cex.main = 2)
R> title("\n\nk=6 Nearest Neighbor Graph")
```

The preprocessing for this example included generating a $k = 6$ nearest neighbor graph using the Euclidean distance metric and applying Floyd's algorithm to the graph (refer to the right panel of Figure 4 for the k-NN graph). These settings captured the manifold. For classification, we computed the true unlabeled classification accuracy averaged over 50 simulations. For this, the labeled size varied among 5%, 10%, 50%, and 90%. The results of the simulation are given in Table 2.

	5%	10%	50%	90%
SPA	0.766 ± 0.006	0.873 ± 0.004	0.950 ± 0.006	0.964 ± 0.002
supervised k-nn	0.675 ± 0.005	0.805 ± 0.005	0.935 ± 0.001	0.953 ± 0.002

Table 2: Accuracy rates for simulation.

The results for the $k = 6$ supervised nearest neighbors algorithm was also included for comparison purposes. For these data, we see that **spa** outperforms the supervised k-nn especially when the $\mid L \mid$ is small. A single run with 10% labeled is given below:

```
R> g6 <- knnGraph(x, k = 6)
R> Dij <- floyd(g6)
R> set.seed(100)
R> L <- sample(1:n, ceiling(0.1 * n))
R> U <- setdiff(1:n, L)
R> y1 <- y
R> y1[U] <- NA
R> gself <- spa(y1, graph = Dij)
R> tab <- table(fitted(gself)[U] > 0.5, y[U] > 0.5)
R> sum(diag(tab)) / sum(tab)

[1] 0.72
```

6.2. Protein interaction data set

For this next example, the observations consist of 1875 proteins from yeast organisms in the form of a graph. For this particular data 13 different systems are used to detect interactions between any two-proteins, e.g., a protein interaction could be detected by two-hybrid, immunoprecipitation, affinity-purification, etc (Ruepp *et al.* 2004). The edges for this graph are defined by the count of interactions detected from different systems, i.e., e_{ij} is given by:

$$e_{ij} = \sum_s \mathbf{1}_{\{\text{system } s \text{ detects interaction between protein } i,j\}},$$

It is assumed that each system would detect an interaction between two identical proteins, i.e., $e_{ii} = 13$ for all i. The response for this data set is the classification of whether the protein is located on a nucleus of the cell. For these data, we do not have the response for 685 observations. The goal is to use the protein interaction network to classify the cell location for these observations. The following R code reads in the data:

```
R> data("protLoc")
R> y <- protLoc$class
R> gs <- protLoc$inter
R> A <- gs[[1]] + gs[[2]]
R> diag(A) <- 13
R> resp <- as.data.frame(sapply(1:nlevels(y),
+    function(i) as.numeric(y == levels(y)[i])))
```

```
R> names(resp) <- levels(y)
R> y <- resp[, 20]
R> n <- length(y)
```

Next we fit the SPA to these data. To assess performance we generate a training set and a tuning set with 50% of the non-missing cases in both.

```
R> set.seed(100)
R> L <- sample(which(!is.na(y)), 0.5 * n)
R> tuneind <- setdiff(which(!is.na(y)), L)
R> uind <- which(is.na(y))
R> U <- c(tuneind, uind)
R> y1 <- rep(0, n)
R> y1[L] <- y[L]
R> y1[U] <- NA
R> g <- spa(y1, graph = A, control = spa.control(diss = FALSE))
R> tab <- table(y[L], fitted(g)[L] > 0.5)
R> a1 <- sum(diag(tab)) / sum(tab)
R> tab <- table(y[U], fitted(g)[U] > 0.5)
R> a2 <- sum(diag(tab)) / sum(tab)
R> a3 <- sum(apply(A[U, L], 1, sum) == 0) / (length(U)) * 100
R> cat("Labeled Acc.=", round(a1,4), " \nUnlabeled Acc.", round(a2, 4),
+    " \n% zero A_ULx 1=", round(a3, 4), "\n")

Labeled Acc.= 0.9989
Unlabeled Acc.= 0.77
% zero A_ULx 1= 41.0448
```

From these results, we notice that the labeled accuracy is 0.999 and the unlabeled accuracy on the tuning set is 0.77. In addition, 41% of the unlabeled cases have no direct link to the labeled response. Updating using transductive prediction addresses this issue and can improve the fit. We use the tuning set to optimize the SPA regularization parameter settings for ℓ and k. In the case of ties, we choose the smallest k and largest ℓ to yield the least regularization.

```
R> levs <- c(0, 5, 10, 100, length(U))
R> lss <- c(seq(0, 100, length = 7), Inf)
R> tunes <- matrix(0, length(levs), length(lss))
R> t1 <- proc.time()
R> for(j in 1:length(levs)) {
+    for(i in 1:length(lss)) {
+      g1 <- update(g, ynew = y1, gnew = A, dat = list(k = j, l = lss[i]))
+      tab <- table(g1[U] > 0.5, y[U])
+      tunes[j, length(lss) - i + 1] <- sum(diag(tab)) / sum(tab)
+      cat("j=", j, " i=", i, "time=", (proc.time() - t1) / 60, "\n")
+    }
+  }
```

```
j= 1  i= 1 time= 0.01533333 0.003016667 0.01075 0 0
j= 1  i= 2 time= 0.0302 0.00535 0.02081667 0 0
...
j= 7  i= 4 time= 0.6144167 0.1131833 0.4250333 0 0
```

```
R> tlev <- unique(apply(tunes, 1, which.max))
R> tls <- apply(tunes[,tlev], 2, which.max)
R> i1 <- which.max(diag(tunes[tls, tlev]))
R> olev <- tls[i1]
R> otls <- tlev[i1]
R> gprot <- update(g, ynew = y1, gnew = A, dat = list(k = olev, l = otls),
+    trans.update = TRUE)
R> tab <- table(y[L], fitted(gprot)[L] > 0.5)
R> a1 <- sum(diag(tab)) / sum(tab)
R> tab <- table(y[U], fitted(gprot)[U] > 0.5)
R> a2 <- sum(diag(tab)) / sum(tab)
R> cat("Labeled Acc.=", round(a1, 4),
+    " \nUnlabeled Acc.=", round(a2, 4), "\n")
```

```
Labeled Acc.= 0.9989
Unlabeled Acc.= 0.78
```

From this result, we notice some improvement with the SPA over the FTF algorithm directly. Using this SPA fit we then get the following as the predictions for the unlabeled responses.

```
R> round(fitted(gprot)[uind],4)
```

```
[1] 0.3725 0.4493 ... 0.6312 0.1688
```

7. Concluding remarks

In this paper, an R package **spa** that implements a semi-supervised framework for fitting semi-parametric models was presented. Its key features are parameter estimation in semi-supervised learning, the distinction between transductive and inductive prediction, and a variety of functions for processing a graph.

References

Belkin M, Matveeva I, Niyogi P (2004). "Regularization and Semi-Supervised Learning on Large Graphs." In *COLT*, pp. 624–638.

Blum A, Chawla S (2001). "Learning from Labeled and Unlabeled Data Using Graph Mincuts." In *International Conference on Machine Learning*, pp. 19–26.

Blum A, Mitchell T (1998). "Combining Labeled and Unlabeled Data with Co-Training." In *Computational Learning Theory*, pp. 92–100.

Chapelle O, Schölkopf B, Zien A (eds.) (2006). *Semi-Supervised Learning*. MIT Press, Cambridge.

Culp M, Michailidis G (2008a). "Graph-Based Semisupervised Learning." *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **30**(1), 174–179.

Culp M, Michailidis G (2008b). "An Iterative Algorithm for Extending Learners to a Semi-Supervised Setting." *Journal of Computational and Graphical Statistics*, **17**(3), 545–571.

Eppstein D, Patterson M, Yao F (1997). "On Nearest Neighbor Graphs." *Discrete and Computational Geometry*, **17**(3), 263–282.

Joachims T (2003). "Transductive Learning via Spectral Graph Partitioning." In *International Conference on Machine Learning*, pp. 290–297.

Kondor R, Lafferty J (2002). "Diffusion Kernels on Graphs and Other Discrete Input Spaces." In *International Conference on Machine Learning*, pp. 315–322.

Koprinska I, Poon J, Clark J, Chan J (2007). "Learning to Classify E-Mail." *Information Science*, **177**(10), 2167–2187.

Kui M, Zhang K, Mehta S, Chen T, Sun F (2002). "Prediction of Protein Function Using Protein-Protein Interaction Data." *Journal of Computational Biology*, **10**, 947–960.

Little R, Rubin D (2002). *Statistical Analysis with Missing Data*. 2nd edition. John Wiley & Sons, New York.

McCallum A, Nigam K, Rennie J, Seymore K (2000). "Automating the Construction of Internet Portals with Machine Learning." *Information Retrieval Journal*, **3**, 127–163.

Nui Z, Ji D, Tan C (2005). "Word Sense Disambiguation Using Label Propagation Based Semi-Supervised Learning." In *Association for Computational Linguistics*, pp. 395–402.

R Development Core Team (2011). "R: A Language and Environment for Statistical Computing." ISBN 3-900051-07-0, URL http://www.R-project.org/.

Ruepp A, Zollner A, Maier D, Albermann K, Hani J, Mokrejs M, Tetko I, Guldener U, Mannhaupt G, Munsterkotter M, Mewes H (2004). "The FunCat, a Functional Annotation Scheme for Systematic Classification of Proteins from Whole Genomes." *Nucleic Acids Research*, **32**(18), 5539–5545.

Tenenbaum J, Silva V, Langford J (2000). "A Global Geometric Framework for Nonlinear Dimensionality Reduction." *Science*, **290**(5500), 2319–2323.

Wang F, Zhang C (2006). "Label Propagation through Linear Neighborhoods." In *International Conference on Machine Learning*, pp. 985–992.

Zhan F, Noon C (1998). "Shortest Path Algorithms: An Evaluation Using Real Road Networks." *Transportation Science*, **32**, 65–73.

Zhou D, Schölkopf B, Hofmann T (2005). "Semi-Supervised Learning on Directed Graphs." In *Advances in Neural Information Processing Systems 17*, pp. 1633–1640. MIT Press, Cambridge.

Zhu X (2008). "Semi-Supervised Learning Literature Survey." *Technical report*, Computer Sciences, University of Wisconsin-Madison.

Zhu X, Ghahramani Z, Lafferty J (2003). "Semi-Supervised Learning Using Gaussian Fields and Harmonic Functions." In *International Conference on Machine Learning*, pp. 912–919.

A. Large graph considerations

The **spa** package is designed to process a graph of approximately 5000 nodes or less. In the case of graphs constructed from X data, and several observed graphs this is a reasonable restriction. The software can handle larger graphs in batch mode but is unlikely to work on a significantly larger graph. In this appendix, we provide considerations for processing a graph too large for the current software. In this capacity, we assume that the graph is large, unweighted, irreducible and that the labeled size (response set) is small.

For an example of a large graph, suppose that the lattice in Figure 1 grew to be 1000×1000. To process a graph of this size, one could simplify the SPA algorithm and perform a generalization of the result in Figure 1. The technical details are somewhat involved but the idea operates similar to the example in the figure. To start, process the graph into vertex subsets where $\mathcal{V}_0$ are the labeled cases, $\mathcal{V}_1$ are the set of vertices one hop away from the labeled cases, ... and $\mathcal{V}_{k^\star}$ are the vertices $k^\star$ hops away. For SPA at iteration r, we have that $\mathcal{V}_r - \mathcal{V}_{r-1}$ are the vertices to be labeled, $\mathcal{V}_{r-1}$ are the vertices labeled from previous rounds, and $\mathcal{V}_0$ contains the observed labeled cases. Denote $y = Y_{\mathcal{V}_{r-1}}$ as the vector with true labeled cases for observations in $\mathcal{V}_0$ and the final estimates from previous rounds for the other cases. From this, one can apply FTF at iteration r which reduces to: $\hat{Y}_i = 0$ for all $i \in \mathcal{V}_r - \mathcal{V}_{r-1}$ then iteratively fit:

$$\hat{Y}_i = \tfrac{1}{D_i} \sum_{j \in \mathcal{V}_{r-1}} W_{ij} y_j + \tfrac{1}{D_i} \sum_{j \in \mathcal{V}_r - \mathcal{V}_{r-1}} W_{ij} \hat{Y}_j \tag{8}$$

where D_i normalizes each case, and W_{ij} reduces to the indicator that observation j is in the neighborhood of i since the graph is unweighted. Once this converges then we penalize $\hat{Y}$ towards the prior with an increasing penalty. For example, let π_1 be the prior for class 1, then we set

$$\hat{Y}_i = \hat{Y}_i + \left(\tfrac{r}{1+r}\right)^\gamma \left(\pi_1 - \hat{Y}_i\right). \tag{9}$$

The estimates for class 0 are obtained in a similar manner. The estimates are treated as truth for the next round. The last step is to adjust the labeled estimates as $\hat{Y}_L = QT_{LL}Y_L + (I - Q)S_{LU}\hat{Y}_i$ (refer to Section 4.2). The most computationally demanding step is in the construction of the vertex subsets and special considerations should be considered depending on the graph (Zhan and Noon 1998).

In the above algorithm, (8) uses the Nadaraya Watson line estimator form of kernel regression to generate PCEs as opposed to the **spa** package which uses the matrix $S = D^{-1}W$. In the convergent solution both are equivalent. For a small graph computing the matrix approach is fast and requires the entire graph in memory. However, for a large graph computation using (8) is more practical since we only need the local observations to the vertex being estimated (i.e., W_{ij} is zero for most vertices in the expression).

For (9) we are penalizing the estimates towards the class prior to account for observations farther away from the labeled cases. The penalizing function $f_\gamma(r) = \left(\tfrac{r}{1+r}\right)^\gamma$ is a cumulative density function. For the package SPA we estimate the cumulative density function from the data using both the parameter k and ℓ in the `dat` argument as described in Culp and Michailidis (2008a). Estimating this function directly for a large graph is computationally demanding, but using $f_\gamma(r)$ in (9) is simpler and likely sufficient. For γ, values too small will

result in classification of several PCEs as the prior while large values will diminish the role of the local information in classification. To estimate γ one could use a tuning set similar to the protein interaction example in Section 6.2. To estimate $k^\star$ notice that (8) moves towards the prior for r large enough and thus one could estimate using (9) until the PCEs are sufficiently close to the prior, i.e., the estimates resulting from round $k^\star$ are such that $\hat{Y}_i \approx \pi_1$. The remaining nodes would then be estimated directly as the prior.

Affiliation:

Mark Culp
Department of Statistics
West Virginia Univeristy
PO Box 6330
Morgantown, WV 26506, United States of America
E-mail: mculp@stat.wvu.edu
URL: http://www.stat.wvu.edu/~mculp/

Journal of Statistical Software http://www.jstatsoft.org/
published by the American Statistical Association http://www.amstat.org/
Volume 40, Issue 10 *Submitted:* 2010-07-23
April 2011 *Accepted:* 2011-01-11